ISI Research Report
ISI/RR-88-203
July 1988

University of Southern California

Robert Neches

FAST Workstation Project Overview

INFORMATION SCIENCES INSTITUTE
213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

AD-A197 924

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT This document is approved for public release, distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) ISI/RR-88-203	5. MONITORING ORGANIZATION REPORT NUMBER(S) --------------
6a NAME OF PERFORMING ORGANIZATION USC/Information Sciences Institute; 6b OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION --------------
6c ADDRESS *(City, State, and ZIP Code)* 4676 Admiralty Way Marina del Rey, CA 90292	7b. ADDRESS *(City, State, and ZIP Code)* --------------
8a NAME OF FUNDING / SPONSORING ORGANIZATION DARPA; 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA903-86-C-0178
8c. ADDRESS *(City, State, and ZIP Code)* DARPA 1400 Wilson Blvd. Arlington, VA 22209	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO -------------- ; PROJECT NO. -------------- ; TASK NO. -------------- ; WORK UNIT ACCESSION NO --------------

11 TITLE *(Include Security Classification)*
FAST Workstation Project Overview [Unclassified]

12 PERSONAL AUTHOR(S) Neches, Robert

13a TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Research Report	FROM ________ TO ______	1988, July	28

16 SUPPLEMENTARY NOTATION

17 COSATI CODES

FIELD	GROUP	SUB-GROUP
09	02	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
expert systems, intelligent interfaces, procurement, workstations. (KR)

19 ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The FAST Workstation and the FAST Broker projects are companion efforts, which jointly seek to demonstrate a model for the pursuit of electronic commerce. This model is being instantiated in a system supporting procurement of standard electronic parts at low cost with short lead times, which seeks to provide a useful purchasing and information service to DoD and the DARPA VLSI research community. The FAST Broker project focuses on utilization of rapid electronic networks to provide an intermediary that speeds communications between buyers and vendors. The FAST Workstation project focuses on the development of user and software interfaces to enable human participants in the process to easily integrate information and engage in transactions with the system. This overview of the FAST Workstation project is divided into four parts. Section 1 describes the research goals of the effort. Section 2 discusses the relationship between those research goals and the somewhat more applied goals of the overall FAST effort. Section 3 reviews our research approach in the four major activity areas of the project. Section 4 summarizes accomplishments to date. Keywords:

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] [✓ SAME AS RPT] ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Sheila Coyazo Victor Brown	22b. TELEPHONE *(Include Area Code)* 213-822-1511; 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR   83 APR edition may be used until exhausted
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

ISI Research Report
ISI/RR-88-203
July 1988

University of Southern California

Robert Neches

FAST Workstation Project Overview

INFORMATION SCIENCES INSTITUTE
213/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

This research is supported by the Defense Advanced Research Projects Agency under Contract No. MDA903-86-C-0178. Views and conclusions contained in this report are the author's and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any person or agency connected with them.

1. Goals of the FAST Workstation

The immediate goals of the FAST Workstation project are: (1) to provide a package of flexible, generic software tools; and (2) to demonstrate how particular users of the FAST Broker can customize those generic tools to create friendly interfaces to the Broker that simultaneously meet specialized needs and requirements of their organization. These immediate goals exist within the context of a larger research goal: the development of a methodology to facilitate construction of *integrated user support environments.* These environments offer non-programmers what programming environments offer computer programmers -- a set of software tools that work together to help users accomplish the bulk of their daily activities. Our society is moving toward "paper-less" work environments in order to remedy pressing -- and expensive -- problems stemming from the need to manage large amounts of information. As we do so, our ability to provide such environments will be increasingly tested.

Just as a programming environment has identified a set of tools (structure editors, compilers, debuggers, etc.) that assist in the goals of the programming domain, the Fast Workstation project seek to identify analogous general-purpose tools that assist in domains concerned with the manipulation of technical and logistical information. Examples of such tools include mail systems, calendar and scheduling systems, advisory expert systems, management information systems, personal and public databases, knowledge bases, and their accompanying browsing and retrieval aids. Just as structure editors provide a customizable tool in programming environments because a language-independent editor can be designed to interpret externally-provided syntax specifications, we would like our tools to be built in a general-purpose way that enables them to be tuned for specific applications. Just as good programming environments are marked by close integration between their tools (for instance, the ability to enter an editor to examine source code relevant to one's state in a debugger), we would like the organization of our tools to facilitate the smooth transfer of information and control between them. For example, we have demonstrated integration between a mail system and a database browser such that when the mail system is invoked from the browser, users see message templates that are already partially completed based upon the data under examination, and can recursively invoke the browser on either the same or different databases to obtain information needed for remaining fields.

In other words, the FAST Workstation project has the dual roles of:

1. demonstrating tools that users could draw upon to facilitate interaction with the FAST Broker; and

2. researching the notion of "user support environment shells" which, in analogy to expert system shells, provide a framework that facilitates construction of specific information management systems.

This report expands upon these two roles. Section 2 discusses the first role with an emphasis on our analyses of the needs and requirements of potential government users of FAST, such as the Defense Logistics Agency. We focus on describing user problems requiring assistance, because the workstation capabilities addressing those needs are described as part of the research. Section 3 discusses the second role, which is primarily concerned with exploring how to use classification-based declarative knowledge representations as a unifying paradigm to enhance both the usability and modifiability of packages of related software tools intended to run on AI workstations.

2. The Workstation's Role in the FAST Concept

The overall goal of FAST is to greatly accelerate progress towards electronic commerce in the course of developing an automated brokering system for rapid procurement of standard items (demonstrated in the area of standard electronic parts). The system uses computerized network communications and intelligent workstations to significantly decrease the lead time required for the purchasing process.

DARPA is motivated to advance the concept of computerized commerce because of the great benefit it can provide to the DARPA research community, the DoD development laboratories, and to DoD procurement in general. There is also the possibility that this effort could spark a much wider effort in the country to effectively utilize computers in commerce and greatly increase the nation's productivity and competitiveness in the world marketplace.

2.1. Value Added by the FAST Workstation

The Fast Workstation project's primary applied goal is to provide a friendly interface that will assist users in preparing messages that meet the requirements of the Broker. It will also help users reason about how to formulate requests that best meet their own needs. For example, when requesting price quotes for guidance in making design choices to minimize cost, a user would like to ensure that the request covers all and only those alternatives with the particular properties critical to the design.

The FAST Workstation will provide an intelligent agent that blends technical expertise and purchasing expertise in a single location. This will help reduce delays in administrative lead time that currently arise because of communications delays and misunderstandings between the technical people who know what they need and the administrative people who know how to purchase items, but do not recognize the significance of the items they are ordering. Lack of expertise can cause purchasing personnel to fail to detect inappropriate substitutions, to lose time clarifying ambiguous or invalid requests, and so on.

The Workstation will also record policies and procedures, explain them to users, and monitor the sequence of activities in each purchasing process to ensure compliance with applicable procedures and regulations. For example, we have learned that at the Defense Logistics Agency (DLA) management would like to assign purchase requests to a buyer on the basis of a (currently implicit) profile of the kinds of activities the buyer is capable of handling. Factors that management would like to take into account in making these assignments include the buyer's current workload, the technical complexity of the item, and the contracting issues that arise from the size and price of the intended purchase (for example, different regulations apply to purchases under $25,000 than to those over that amount).

DLA does not have the technology needed to record the assumptions underlying an assignment, to ensure that those assumptions are not violated as the process evolves, or to recommend appropriate action if they are. A common occurrence, for instance, is that a second purchase request is initiated for the same item before the buyer completes handling of the original purchase request. For a number of reasons, the second request may not be assigned to the same individual, although in current practice the fact that it exists is flagged on the original buyer's terminal. Current practice also mandates that the two buyers should communicate to agree upon the handling of the requests; there are regulations governing the circumstances under which the buyers are obligated to combine them into a single, larger purchase or may continue to handle them separately.

In addition to orchestrating the buyer-to-buyer contact, by collecting the information needed to decide about combining, it would be desirable for the Workstation to notice when the effects of doing so raise issues with respect to management's profile for the user. If combining the two purchase orders raised the expected cost of an order over $25,000, for instance, more complex contractual issues are raised. Depending on circumstances and the point of progress in the purchasing process when the problem arises, it might be appropriate for the Workstation to advise the buyer to request reassignment of the task to a more experienced buyer. If not, it might be possible for the Workstation to advise on what further issues must be taken into account and what work done so far can be reused (e.g., the buyer might still be able to use some price quotes previously obtained because quotes are often provided with respect to quantity ranges, but might need additional quotes to meet more stringent requirements concerning competition).

2.2. Relation of Applied Goals to Research Goals

The kind of support just described involves guiding the user on the content, order, and timing of the activities required for purchases in particular categories. Workstation support of this kind will reduce administrative lead-time delays caused by failures to follow policy (e.g., protests by would-be suppliers, requests for further documentation or outright rejection during competition advocacy review, etc.) We will discuss how this sort of support would be implemented in the Workstation later in Section 3.3. It is important here, however, to note that providing this kind of support is difficult under conventional technology, and that providing such capabilities in an automated environment requires research. There are several factors in the scenario just described which challenge software developers:

- There is dynamic reasoning required at run time. The behavior of the Workstation has to be sensitive to the particular constraints specified by management for the given buyer. These combine several different factors, and profiles for different buyers potentially involve different combinations of parameters for each factor. Thus, software developers are faced with an irritating combinatoric problem if they seek to algorithmically specify in advance the workstation's handling of any possible profile.

- There is a requirement for monitoring across a broad range of user activities. This is a challenge for software developers, regardless of whether specifying their system in algorithmic fashion or using a rule-based expert system, because it threatens the modularity of a system design -- and therefore the maintainability of that system. The software developer is forced to carefully analyze the functionality of the system in terms of all of the points where an action can be taken that might violate any of the constraints, to ensure that branches are taken (or rules are invoked) that check those constraints *at those points.* This means that code for handling those constraints has to be sprinkled throughout the software. Ensuring that the "sprinkles" fall at every place they should -- and that they continue to do so as the system is maintained over the years -- becomes a daunting task as new constraints are specified and/or new capabilities are added to the system.

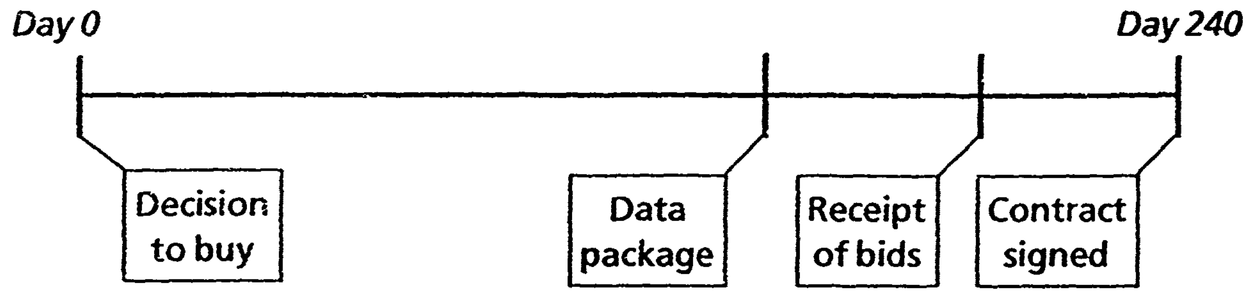
Although it is possible today to build systems which will do the job called for in the scenario above, the task is difficult and the resulting systems are hard to extend and maintain. In our approach, we are trying to provide a framework that makes it easier for software developers to create and maintain user support environments for tasks of this nature. In doing so, we are seeking to follow in the footsteps of software constructors in other areas (e.g., expert system shells and database management systems), by providing specification languages and processing capabilities that prepackage for implementors application-independent aspects of a certain class of software. In the discussion of our research approach in Section 3, we will describe our effort to address such problems and the supporting technology upon which our approach is based.

2.3. DoD-related demonstrations

Since the Workstation project is interested in customizing the general-purpose tools it is building in order to demonstrate their utility in realistic DoD procurement system scenarios, an ongoing activity during the project has involved meetings with a number of DoD agencies to study current procurement processes and computer support. These have included multiple meetings with Air Force representatives from procurement groups at Wright-Patterson AFB and Hill AFB, with Naval representatives involved in automation efforts such as the APADE project, and most recently, with representatives of the Defense Logistics Agency at both the Headquarters and supply center levels.

The Defense Logistics Agency has expressed strong interest at the headquarters level in becoming a participant in the FAST Broker experiment and in developing a technology transfer path from the FAST Workstation to its own **DPACS** system; which is currently under development. Figures 1 and 2 illustrate the issues driving DLA's concern with support environments for their procurement agents. In order to

Figure 1 Problems in Procurement: Adminstrative Lead Time



Average administrative lead time = 240 days

- **Large number of decisions:**
  800 identified in a study of a particular commodity

- **Small percentage really require human judgement:**
  E.g., Defense Logistics Agency estimates that 68% of adjustments in quantity ordered are due to rounding to standard lot sizes

Bottlenecks

- **Regulations**

- **Gathering information used in decisions**

- **Slow communications with the outside world**

Figure 2
Hidden Penalties of
Conventional Procurement

Long lead times cause higher inventory costs

- Defense Supply Centers must maintain an average of $1.5 billion in parts on hand, just in order to ensure timely availability; "just-in-time" ordering is difficult

- Requirements must be forecast on a schedule which takes lead time into account:

  longer-range forecasts = less accurate forecasts

Limited information links and heavy workloads hamper buyers searching for best prices

- Contracting offices making 5000 buys/month are common; a buyer may have up to 300 purchase requests active

- Even after suppliers are identified, time spent awaiting their responses to specific inquiries may exceed internal processing time by factors of 10:1

- These constraints discourage extensive shopping

- Suppliers have reduced incentive to keep their prices low make our description of the Workstation effort more concrete and to illustrate its potential technology transfer opportunities to DoD, we have tended to cast our examples in terms of DLA's problems and needs[1]. DPACS (DLA Pre-Award Contracting System) is a prototype syst . that represents a major step for DLA in the direction of a paperless office environment. It is integrated with a management information system that gives supervisors some, albeit limited, abilities to automate the assignment of purchase orders. Individual buyers interact with workstations that allow them to see lists of purchase orders assigned to them, send messages to other offices, format information packets and contracts for mailing to vendors, and inspect and modify different databases (which contain information such as technical characteristics of items, potential sources of supply, and vendor performance records).

In recent months, in order to understand present and planned DLA systems, we have visited DLA operations at Defense Electronics Supply Center (DESC) in Dayton and Defense Industrial Supply Center (DISC) in Philadelphia, and both Defense Construction Supply Center (DCSC) and DLA Systems Automation Command (DSAC) in Columbus. Assuming the continued interest and support of DLA Headquarters, we are expecting to target our demonstrations in the third year of the project to illustrate the possibilities for technology transfer to DPACS and related DLA computer support systems for logistics.

3. Research Approach

The preceding section illustrated the kinds of services that are needed, and the need for research to make it easier for system developers to produce software that provides those services. This section describes the issues and activities the project is pursuing towards the end of meeting those needs. Our research goals center on usability and modifiability.

Enhancing the usability of Workstation software involves two major questions. The first question is how to design tools that help users keep track of extended activities. The second question is how to implement those tools in a manner that maximizes the extent to which knowledge manipulated in one part of the system, be it application code or user interface code, is accessible and used appropriately in all other parts of the system.

Enhancing the modifiability of these systems involves exploring how to center their implementation around a single, uniform knowledge base shared by all components. The research question is how to develop forms for entering the various kinds of knowledge used by the system in such a way as to maximize the usability of

[1]Please note, however, that this is by no means the only possible testbed for the workstation effort. For example, the workstation support environment could be customized in other directions to, say, support designers. The DARPA VLSI research community could benefit from a workstation which, as a design evolved, provided convenient access to information about pricing and availability of parts, coupled with the ability to generate orders for all parts on a bill of materials for the design.

the knowledge representation for multiple purposes. In doing this, we hope to minimize the extent to which conceptually equivalent information is duplicated in different forms in various components of systems.

Under the umbrella of these global research concerns with usability and modifiability, the FAST Workstation project is engaged four specific areas of research: control reasoning, information access tools, activity specification, and interface specification. The following is a brief statement of each of these research areas.

1. Development and augmentation of mechanisms for control of reasoning within knowledge representation languages. Particular concerns include: developing a more explicit linkage between rules and the data upon which they operate, thereby making it easier for system developers to determine which parts of the system are impacted by a modification. These techniques will be applied in construction of knowledge bases for both the Broker and the Workstation.

2. Development of a knowledge base and database browser. Particular concerns include: mechanisms for allowing flexible search criteria, and managing requests that involve querying with several different databases.

3. Development of a representation language (called "Scenarios") for extended activities. Particular concerns include: specifying relationships between sub-activities, providing mechanisms for helping users keep track of the status of various activities, and generating advice about allowable next activities given the current state of progress in a process. The Scenarios language will be used both for specifying protocols for interaction with the Broker, and for specifying procedures and regulations governing various kinds of purchases.

4. Development of specification techniques for knowledge-based user interfaces. Particular concerns include: providing means for designers to explicitly state policies for how system concepts will be presented, ensuring that design policies are implemented consistently throughout, and generating documentation and help from the interface specification. These techniques will be used to develop the user interfaces for Workstation components such as the database browser and an electronic mail interface to the broker.

These areas will be described in more detail in the subsections below. The common unifying thread running through these activities can be summarized as follows.

The unifying goal is to develop a single declarative mechanism that can be used to reason about extended activities in many different ways, so that a single representation can be used to generate forms for presentation, to structure interaction dialogues, to organize history and audit-trail mechanisms, and to drive help and explanation facilities, and so on. There are two ways that developing this representation technology, which is not currently available, would facilitate the process of constructing future workstations for different application domains. First, code size would be reduced because there would be less knowledge that had to be duplicated in slightly differing guises. Second, different workstation capabilities that tap the same underlying knowledge would be more consistent because there would be less risk of introducing the kinds of discrepancies that currently can arise when different capabilities represent equivalent information in different forms. Our approach to these issues involves extending and applying classification-based reasoning capabilities in AI knowledge representation languages.

3.1. Research on Control of Reasoning

3.1.1. Background

ISI has some special technology for addressing these issues. NIKL [Moser 83, Kaczmarek, et al. 86] and LOOM [Mac Gregor 87a] are part of a body of software developed under DARPA auspices that provide significant leverage for our research.) What NIKL has that commercial languages lack is an automatic classifier [Schmolze 83] for analyzing its knowledge base. KEE comes closest to this among commercial languages because its knowledge representation is developed from the same historical roots in KL-ONE; it has better-developed control mechanisms, but no classifier. ISI is extending NIKL to improve its control mechanisms, and will have them plus the advantages of a classifier.

What does a classifier do? In KL-ONE-based languages, knowledge is expressed in syntactic units called "concepts." The NIKL classifier organizes all of the concepts it knows about into a taxonomy in which more specialized concepts are placed below more general ones. When the classifier is presented with a new (fully- or partially-specified) concept, it can determine where that new concept should be located -- i.e., it can relate this new piece of knowledge to the already existing knowledge. Furthermore, the classifier can discover additional properties of that concept description which were implied by the interaction of its description with the knowledge base, but which were not explicitly stated.

The inference capabilities provided by a classifier can be used in a variety of settings. A classifier can be used analogously to a pattern matcher for a rule system, as an information retrieval mechanism, as a policeman of correctness and completeness in augmenting a knowledge base, as a forward- and backward- chaining inference engine, or as a realizer (a mechanism for identifying which concepts describe a given situation or a specific fact). Thus, although a classifier is not a general-purpose reasoner, it does address a number of different significant problems.

The type of reasoning exhibited by a classifier applies to several different problems. For example, in the process of adding new knowledge to a knowledge base (i.e., adding new concepts), a user may fail to explicitly specify all of the information relevant to a particular concept. Often when properties are not explicitly stated, it is because the person or program creating a new concept did not think of them. The

NIKL classifier can deduce knowledge about a concept which was implied but not explicitly stated; thus, NIKL performs a powerful role as a mechanism that discovers otherwise neglected possibilities and brings them to the attention of other reasoning components.

3.1.2. New Work

Current technology enables us to build descriptions of new concepts and determine their appropriate placement in a taxonomic hierarchy of previously defined concepts. We are seeking to add the capabilities to control activity triggered when such a classification occurs, and to use both structural and semantic information contained in the hierarchy to reason about similarities between the newly-added description and closely-equivalent concepts.

In a number of previous papers [Neches, et al. 85, Neches, Swartout, and Moore 85, Swartout and Neches 86], we have argued for the importance of a separate and explicit *terminological space.* By this we meant that the rules or methods of expert systems interact to produce results by creating situations to which other rules or methods respond. The language of terms used to represent such situations comprises the terminological space for a system. If the terms are not independently defined, then they are implicitly defined by the behavior of the code that manipulates them. This creates barriers to the maintainability and explainability of systems: there is no way to ensure that terminology is used consistently across parts of a system, and no way to determine how it was intended to be used. (Clancey [Clancey 83] has also been pointing out these issues for some time.) The Explainable Expert System project at ISI has been somewhat concerned with maintainability, though largely concerned with explanation, and has concentrated on the task of encouraging designers to explicitly specify the abstract plans and goals that lower-level methods or rules are intended to realize.

The FAST Workstation project, in contrast, is focusing explicitly on developing an execution environment. The goal is to promote integration of different software components, as well as maintainability, by making it possible to program directly in terms of actions and goals associated with terminology modeled explicitly in LOOM. In contrast to the traditional *recognize-act* cycle of rule-based systems, the cycle of our architecture might be characterized as:

- *realize* -- LOOM descriptions of data objects are formed, and LOOM is used to determine where those descriptions would fit with respect to concepts in the knowledge hierarchy;

- *inherit* -- on the basis of the LOOM classifications, goals or actions associated with concepts that describe the data are collected;

- *select* -- the set of active goals is augmented by the new-found set, then pruned;

- *plan-and-act* -- the system generates sub-goals and new data objects in response to its set of active goals, its knowledge base is modified accordingly, actions which affect the environment outside the system are executed, and the cycle is primed to repeat.

3.1.3. Relation to Other Projects

This architecture is being developed in collaboration with ISI's Knowledge Representation project [Mac Gregor 87b], which is developing the facilities for realization and inheritance. The FAST Workstation effort is developing the selection and planning facilities, as well as providing the applications that serve as testbeds. The general reasoning paradigm is one that has been implemented successfully in the past in application-specific ways in systems such as the CONSUL mapper [Mark 81] and the EES program writer [Neches, et al. 85]. We hope to extend that experience to provide a general facility for programming in this fashion.

3.1.4. Benefits

The environment will support sharing of knowledge between different components of a system, because the LOOM classifier will ensure that declarations of terms with equivalent semantics will map to the same concepts in the knowledge base. We believe that, unlike alternative programming approaches, integration will not be bought at the expense of modularity. This is because programmers will be able to work in terms of the concepts relevant to actions, rather than having to determine the actions relevant to concepts in order to define or modify part of a system's behavior. Herein lies our explanation of how we expect to deal with the kind of monitoring problem described in Section 2.1, where we wished for a system that would notice when any action violated constraints on the buyer specified by management, but wanted to avoid the software development headaches of explicitly programming-in checks for those constraints at every possible action-point. In the architecture we are developing, the constraints circumscribing purchasing tasks allowed an individual buyer would be specified as concepts in the knowledge base. Associated with those concepts would be goals pertaining to responses. As the realize-inherit-select-plan cycle of the architecture proceeds, if at any time the buy in progress violates a constraint, its description would become classified as an instance of a violation and inherit the response goals specified. Although this sounds like what would be possible in a conventional rule-based system, the same effect there could only be achieved by surrendering modularity. This is because a pattern-matcher weaker than the LOOM classifier would not be able to automatically detect that the violation had occurred; therefore, it would be necessary to program in control knowledge telling the system when to check.

3.1.5. Plan of Attack

To ensure useful intermediate results in the FAST domain, our plan of attack has been to implement application-specific classification-based programs in parallel with the design and development of the general classification-based programming architecture. These application-specific programs will be reimplemented in our architecture when it becomes operational. The first realization of this plan has been an implemented program called the Generic Description Generalizer, which has formed the domain-independent basis upon which we have implemented an expert system called the Parts Advisor. This expert system accepts customer-supplied part numbers that do not correspond to an actual existing part, and supplies explanations for possible sources of the error along with suggestions about alternative part numbers that might satisfy the customer's intents.

3.2. Research on Information Access Tools

3.2.1. Background

Our second major activity area examines paradigms for interacting with systems, particularly where information retrieval is concerned. Access to multiple databases, providing information about part descriptions, substitutability of parts, manufacturers and suppliers, etc., is a ubiquitous requirement of the procurement domain. An additional complicating factor is the lack of sophistication among the potential user community; procurement agents are not the facile computer users found in, say, the MOSIS user community. (See [Tomovich 88] for an overview of the MOSIS service.) Ideally, users should be protected from having to know a great deal about the query language and internal organization for each of the different databases they must access. Over and above these usability concerns, there are a number of issues affecting the development and maintenance of user support environments interfacing with multiple databases. We are particularly concerned with avoiding duplication of information between databases and the knowledge bases of expert systems, in order to minimize the cost and effort of maintaining consistency between them.

3.2.2. New Work

To address some of these concerns, we are developing a multi-purpose browsing interface called BACKBORD (Browsing Aid Complementing Knowledge Bases OR Databases) [Yen 88, Yen, et al. 88]. BACKBORD currently operates in two modes. In one incarnation, it acts as an intelligent database interface guided by a NIKL or LOOM knowledge base representing abstractions that map to the conceptual schema of a database. In its other incarnation, it acts as interface to NIKL or LOOM itself. In its latter use as an aid for examining and modifying knowledge bases, BACKBORD provides capabilities for retrieving concepts by description as well as by name, for browsing in an undirected fashion, for organizing systematic inspections, and for adding new concepts to the knowledge base.

BACKBORD's presence in a user support environment is intended to benefit both end-users and system developers/maintainers. To end-users, it provides a uniform interface for retrieving information from all of the information systems in the environment regardless of whether those systems rely on conventional database technology or AI knowledge representations. To system programmers, it provides a mechanism for inspecting the data structures of the environment, for determining where to encode intended modifications, and for identifying the parts of the system potentially affected by those changes.

The genesis of BACKBORD is the "retrieval by reformulation" paradigm for database interfaces pioneered in Xerox PARC's RABBIT system [Tou, *et. al.* 82]. This approach was originally developed to help formulate queries for database users who were capable of recognizing when they had found the data that they needed, but who were not capable of formulating a query to retrieve that data (either because of uncertainty about their needs, or because of unfamiliarity with the database schema or query language). RABBIT allowed users to converge upon a satisfactory query by successive approximation. Users were presented with an initial query, a display of an example item retrieved by the query, and a list of other items retrieved. The system enabled them to examine the example and modify the query to include or exclude features of the example, according to whether those features were desired in the data being sought.

BACKBORD extends the retrieval by reformulation paradigm in a number of important directions. The notion of queries is generalized to *descriptions*, which are built on top of the NIKL/LOOM knowledge representation language and therefore have greater expressive power. BACKBORD provides information about the organization of the knowledge base pertaining to components of both the description and the examples. Refinement of the description is speeded by the way it allows access to related concepts of components, e.g., users can include or exclude not just features from an example, but generalizations, specializations, or sibling concepts from either the example or the description itself. Rather than forcing users to operate in a strictly query-oriented, text-based fashion, BACKBORD combines forms-based textual displays with graphical displays of knowledge networks, allowing the user to work in whichever mode is more perspicuous for a particular task or a particular knowledge base structure. BACKBORD also differs from previous retrieval-by-reformulation systems in being designed for integration with other systems, rather than as a stand-alone data retrieval system.

3.2.3. Work Plan

Having developed an initial system, and demonstrated its ability to map its descriptions into SQL queries, we are now extending BACKBORD in several different directions. Among these are its extension as a general interaction paradigm, and the problem of dealing with multiple, heterogeneous databases. The former is discussed in some depth in [Yen 87]; the key novel idea is that any interaction in which users require help or information in filling out forms can be treated as an instance of the retrieval-by-reformulation paradigm. We refer to this as *specification-by-reformulation*, since the purpose is no longer to flesh out a template for a query, but rather to flesh out a template for some arbitrary form where the system's help and advice is showing the user what the possibilities and constraints are in filling out that form. We are exploring this approach to provide our mail interface to the Broker, for example. A declarative NIKL/LOOM specification of the protocols for different messages to the Broker suffices to guide the system both in creating forms for the user to fill out, in preparing partially filled-out forms when the user wants to contact the Broker about items found while inspecting a database on electronic parts, and in telling the user how to complete the message before it is sent. This serves several goals in our research plan: it is both another testbed application for the classification-based programming environment, and a general facility to include as part of the shell we seek to provide that will make it easier for programmers to rapidly construct user support environments.

Multiple, heterogeneous databases present a second important concern for BACKBORD. It is important to deal with them efficiently and transparently. Although we have demonstrated the ability to generate queries to a single SQL database, and expect soon to demonstrate the ability to access databases remotely, a practical system will not have the luxury of decreeing that it will only access one SQL database. We would like BACKBORD to enable users to access information from different databases. Further, we would like users to be able to do so without needing to know a great deal about which database contains the information, how the different databases are organized, or which DBMS they are implemented within. Our approach is to create a data retrieval interface centered around a knowledge base that contains a database-independent description of the information being manipulated, along with a set of mapping rules indicating how to transform this description into the schema of each individual database. These ideas are being tested by extending the interactive database interface that we have already implemented. This system currently acts as an interface to a single database at a time; the extensions being designed are intended to allow it to support queries that require integrating information across different databases.

3.3. Research on Activity Specifications: Scenarios

3.3.1. Background

The third research activity of the project concerns the development of tools to assist users with time management and activity tracking. These tools will be domain-independent interpreters of task-specific declarations that describe the activities of concern. A key part of this task is defining the language in which those declarations are expressed.

Scenarios are program-like descriptions of the sub-tasks that compose extended tasks such as reading and responding to computer mail. Scenarios are like very high-level procedures, in that they describe a sequence of steps to be performed. However, unlike a procedure, a scenario attempts to capture the processes or sequences of activities that are related by mental plans at the level of the user. Further, a scenario imposes only orderings between steps that are necessitated by dependencies between them.

Agendas are a repository for items representing goals and action requests. An agenda compiles a central record of those activities (current, pending, and past) a particular agent has been designated to perform. Our goal is a language for specifying "scenarios" that denote the responsibilities and actions of various agents in performing tasks, ensuring that specifications of scenarios will include the information that a help system requires.

Scenarios operate by attaching the sub-tasks to assorted agendas, each of which represents the completed and pending tasks of some agent that participates in the task. An important aspect of the notion of scenarios and agendas is their use in providing multiple organizations of history. Events in this scheme are represented as agenda items; the completed events constitute system history. Organized by creation time, these items present a chronological history. Organized by the scenarios that spawned them, they represent a goal-ordered view of events. Organized by the agendas upon which they appear, they represent an attribution-oriented view by providing information on the effects of actions by a particular agent.

Scenarios and agendas provide external memory for "unfinished business", reducing burdens on users' memories. This is extremely important in practical applications of the FAST Workstation. For example, DLA, procurement agents are assigned as many as 300 purchase requests at any given time. It is crucial that they have a mechanism for keeping track of what has been done, what steps are next, and what remains to be done, for each purchase request. Under currently available technology, DLA has been able to build a system that represents the current state(s) of a purchase request (e.g., it can record that bids have been received and are being evaluated with the buyer awaiting responses to questions that had been referred to, say, Technical Operations and Quality Assurance). They have not been able to build a system that has an explicit model of the relationships between those states, which would be able to help users (and their supervisors) with their personal time management by generating pertinent presentations to answer questions like:

- *What should I be working on today?*
- *What should I do next?*
- *What can I do to expedite this request?*
- *What parts of this task have been done?*
- *Where has the time been spent thus far on this activity?*

3.3.2. An Example

Let us consider the following top-level scenario for a small manually processed purchase request. It illustrates the kinds of things that might be captured in scenarios, the kinds of displays users might want generated from the execution of a scenario, and the kind of reasoning that they might expect a system to help them with based on the knowledge contained in a scenario. (It also reveals some of the different kinds of databases that need to be accessed at various points in time.) This scenario has five major steps at the top level:

1. Review the purchase request for completeness

2. Identify potential suppliers, formulate and transmit requests for quotes

3. Review incoming bids for acceptability with respect to the terms of the RFQ

4. Evaluate bids for reasonable price

5. Prepare and send contract award materials to winning bidder.

Each of these five major steps has embedded contingencies that involve alternative sub-scenarios. For example, in the process of identifying potential suppliers, the buyer is required to sample without replacement from a list of qualified vendors. After retrieving that information from a database, ideally, the buyer checks the candidate vendors against various other databases. These provide information about the performance record of the candidates with respect to price, quality, and delivery schedule.

One of the frustrations of the current regulatory environment is that buyers cannot refuse to solicit bids from candidates with poor records. However, the presence of a candidate with a poor record indicates the buyer should act out a special scenario intended to respond to the difficulty. This might include adding additional vendors to the candidate list to increase the likelihood of the poor performer being outbid. It might also include adding conditions to the RFQ or to the final contract, and (to use our jargon) the assertion of extra activities in the agenda of tasks pertaining to analysis of incoming bids. For example, at contract preparation time, the buyer might see an agenda item calling for the inclusion of a clause allowing cancellation if delivery schedules are not met. The general point is that if either the system or the user detects that this special scenario applies, the scheme provides the mechanisms for ensuring that the user is reminded at the appropriate times. This is potentially a very significant aid; keep in mind that procurement agents must deal with literally hundreds of active transactions and these reminders might need to be delivered days, weeks, or even months later.

Answering time management and planning questions of the type described above would become a matter of simply listing agenda items retrieved according to various keys, such as the class of task they represent, the kind of scenario that spawned them, and so on.

3.3.3. Research Plan

Our approach to these goals continues from work done on a plan definition language in the Explainable Expert Systems project [Neches, et al. 85, Swartout and Neches 86]. That language provided a relatively simple syntax for specifying plans in terms of goals that they served to accomplish and methods for achieving them. Goals were represented in NIKL, and a special-purpose planner that utilized the NIKL classifier as a pattern-matcher enabled the system to develop plans on a much more sophisticated basis than merely seeking exact matches between sub-goals in a plan and the capabilities declared for candidate sub-plans. Over and above finding exact matches, the system could find matches based on logical subsumption between the capabilities of a plan and the requirements of a goal, and could also search for reformulations of goals that it could not plan directly. Thus, for our purposes, it provides a way of declaratively describing goal-based activities that has a great deal of flexibility in capturing connections between goals expressed at one level and finer-grained decompositions of those goals at lower levels.

Several extensions and modifications are needed to adapt the plan definition language to the purposes of scenarios. The specification of plan methods assumed an overly-rigid, algorithmic flow of control between the actions in the plan; we need to provide instead a more dataflow-oriented specification since we wish to be able to model alternative action sequences that satisfy a given plan. We need to add operators to the language for creating and modifying agenda items. We wish to move the language from NIKL to LOOM, in order to benefit from the latter language's solutions to a number of limitations in NIKL. Finally, we want to adapt BACKBORD, our general-purpose browsing tool, to serve as an interface to allow users to formulate questions about activities.

The insight behind the work on scenarios is that the sorts of tools which provide this information are generic, with potential benefits in many applications. By providing a domain-independent scenario specification language, and the tools for generating presentations from them, we hope to save system developers a great deal of ground work in creating systems that help their users manage time and track activities.

3.4. Research on Interface Specification

3.4.1. Background

One of the most important attributes of an understandable, usable human-computer interface is *consistency* -- the basic principle that, whenever possible, similar system features ought to look (and operate) in similar ways. The interface construction component of a shell for user support environments ought to actively enforce consistency in the user interface design. However, existing technology for specifying interfaces -- although serving to make consistency possible -- does nothing to require or encourage it. State-of-the-art tools (such as the FORMS-KIT package described in the next subsection) provide the ability to specify display forms in a parameterized, and therefore reusable, fashion. Because the display forms are reusable, an interface designer can use such tools to produce consistent user interfaces by setting and following design policies for when to use each kind of form. However, this is all dependent on the judgment and recall of the designer, because these design policies are implicit, undocumented, and play no formal role in the specification. The designer must ensure that the appropriate form is used at all places where the design policy dictates that it should be used.

For example, in our BACKBORD browsing interface, we have the notion of an *inspect, then select* interaction which is fairly pervasive throughout the system. In this kind of interaction, users are presented with a set of choices, and we would like to let them take action to get more information about some of the choices before selecting a subset of the alternatives. We would like to identify all cases of this sort of interaction and have a policy of presenting them through a *checklist*, a kind of menu in which each item is followed by two boxes, one that displays a checkmark when the user inspects the item, and one that displays a checkmark when the user selects an item. Checklists serve to remind users of what decisions remain to be made in complex selection tasks. The implicit convention is that, when any new facility is added to the system, checklists will be used to present that facility if it satisfies this description.

Our long-range goal is to provide an environment for interface developers that promotes high-quality, consistent designs by making such currently implicit conventions into an explicit component of a formal design specification. Descriptions of our goals for this environment have been published in [Neches 86, Neches 88]. This is a long-term effort, and only a portion of this effort is being pursued under the support of the FAST Workstation project. This subsection briefly describes the overall vision, and the remaining subsections describe the work being performed within the project that pertains to this topic.

The key idea is to build a core knowledge base that describes abstract communication activities at a level common to all interfaces. Given this core knowledge base, we can provide tools that will help interface designers specify their particular interfaces. To do so, they describe the communications of their interfaces; the tools attempt to classify those descriptions as specializations of the generic concepts in the core knowledge base. The designers also define rules for presentation and input analysis, and associate those rules with concepts in the extended knowledge base about communications. These rules are then propagated down by inheritance to the concepts that represent the specific interface design, and the combined overall specification can be checked for conflicts and inconsistencies. In addition to promoting consistency by making it possible to analyze an explicit specification for inconsistency, this approach also seeks to contribute by automating the task of implementing the policies represented in the specification. Presentation properties of communication activities (such as the abstract *inspect, then select* previously described) could be specified at a high level; each specific case that satisfies the description of some abstract concept automatically inherits those properties without the interface builder having to remember to explicitly assign those properties.

This long-range scheme depends on four things:

1. a support environment for designers,

2. a foundation knowledge base,

3. a set of analysis tools for evaluating the consistency of a specification, and

4. a set of implementation aids for generating implementations from a design specification.

The last two areas are not being pursued within the Workstation effort. Since this project focuses on the general problem of user support environments, the tools it produces contribute indirectly to the first goal. A small effort is in progress within this project to contribute to the second area -- construction of a foundation knowledge base. That effort is the topic of the remainder of this section.

3.4.2. Base Technology

The starting point for our work on user interfaces is FORMS-KIT [Kaczmarek 84] [goldman 88], a generalized package for specifying the appearance of forms-oriented presentations (e.g., text displays, menus, and so on). The package is currently in use by a number of DARPA-funded ISI projects, including FSD, Integrated Interfaces, and FAST Workstation. One of its notable features is that it provides a large machine-independent specification base built on top of a relatively small machine-dependent substrate. Thus, interfaces built using FORMS-KIT are likely to be very easy to port across hardware.

FORMS-KIT provides a means of giving declarative descriptions of structured presentations and recording interface output history. The design of the facility includes multi-media devices and provides support for keyboards, mice, text, and bitmap images. The system also provides a methodology for defining the interactive behavior of presentation objects, that is, the interface actions associated with them. Other interaction objects can potentially be built from the primitives that it provides. For example, icons are bitmap images with particular interaction behavior, and menus are textual displays with particular interaction behavior. However, further work is needed before FORMS-KIT supports the full range of presentation mechanisms that would be desirable.

The design of FORMS-KIT also provides an underlying architecture based on data abstraction, intended to help separate code for the user interface from code for the application. A dictionary of defined data types is used to manipulate display objects and invoke application functions.

The limitation of FORMS-KIT is that it expresses the presentation of various communication activities (e.g., the checklist form), without representing the underlying communication activity (e.g., the *inspect, then select* process). This property is representative of the state of the art, not just of the particular system we have chosen to use.

3.4.3. Plan of Action

We would like to remedy two problems with this limitation. First, absence of the specification of the intended use of a form makes it harder for developers inspecting the system to recognize when they should use an existing form rather than creating a new one. Second, it fails to modularize the user interface sufficiently; the interface code that defines a presentation is intertwined with the interface code that manages communications with the application.

Our solution is to develop a layer of language above FORMS-KIT that attempts to better separate the form from the content of presentations. Pedro Szekely's recent thesis on the NEPHEW user interface management system [Szekely 87] has demonstrated the feasibility of this approach, and shows that the modularization problem is solvable. NEPHEW provides the groundwork for making this division by enabling interface designers to modularize the interface into *commands,* which represent the content or intent of a communications activity, and *presenters and recognizers,* which represent the display of that activity. We hope to extend NEPHEW by developing a principled taxonomy of command classes, presenters, and recognizers. This will be based on empirical analyses of the interfaces being constructed for the FAST Workstation. With this taxonomy, we will be able to express generalizations about policies for associating classes of presenters and recognizers with classes of communication concepts. This will allow us to retain the benefits of Szekely's solution to the modularization problem while also offering a solution to the first problem.

4. Accomplishments to Date

Section 3 has described the goals, issues, and plans of action in each of the FAST Workstation project's four research activity areas. In this section, we summarize the results achieved thus far.

Much of the implementation work in the project has centered around a general-purpose information retrieval aid. In the design of this aid, domain-independent components have been developed to operate upon domain-specific specifications. The latter provides an abstraction hierarchy over and above the conceptual schema of a particular database, while the domain-independent components constitute a browsing and retrieval facility. This facility interprets the domain-specific specification in order to provide a great deal of help and guidance to users concerned with understanding the database schema and/or formulating particular queries. The domain-specific specification, represented in the NIKL language, allows an application-builder to associate actions with class concepts defined in the specification. When a user is manipulating any object, the domain specific actions inherited through that object's class memberships are offered to the user as options along with built-in domain-independent functions. Thus, the system readily supports customization for particular applications.

The domain-independent component, BACKBORD, implements a "specification-by-reformulation" paradigm intended to help users who can recognize when they have found the information that they need, but who do not initially know how to formulate a query that will retrieve that information. In this paradigm, BACKBORD presents users with an initial query and an example drawn from the set of items retrieved by that query. It then allows the user to examine the structure of conceptual hierarchies above and below fields in both the query and the example. The system enables the user to utilize the results of that examination to modify the query. Modifications take the form of generalizing or specializing restrictions on either the type of items sought, or on the properties associated with items of a given type.

BACKBORD currently handles databases implemented [illegible] own internal format, or it can generate SQL queries to external databases. I[illegible] a forms- and menu- based interface augmented with a graphical interface. This allows users to select any concept in any part of a query or example, see a graphical depiction of the relationships between that concept and its generalizations and specializations in the knowledge base, and then select one or more of those related concepts for use in modifying the query. A mechanism for "jumping" in the knowledge base (quickly replacing one initial query with another) has been implemented. A history mechanism has been provided, which allows users to return to previous queries without having to reconstruct them.

A domain-specific knowledge base, *FRED* (*F*ast *R*etrieval from *E*lectronics *D*atabases), supplies an abstraction hierarchy above data about memory IC's found in the "IC Master Handbook". (This is a standard reference database widely used in both hardcopy and on-line formats.) Using this knowledge base in conjunction with *BACKBORD* allows users to create arbitrary descriptions of needed items and find parts that satisfy those descriptions. It also enables users to start with a description of a specific part, and to conveniently browse the database to search for close equivalents.

The Workstation also enables users to operate upon part descriptions in order to produce messages concerning them to the FAST Broker system. We have completed the initial implementation on a set of domain-specific actions that will allow users to do this. The implementation is a test of our methodology for supporting readily customizable software. Another technically interesting aspect of that implementation is that the messages and the actions applicable to them are specified in exactly the same format as that used for the information retrieval components of the system. This means that the same general-purpose capabilities in BACKBORD that help users understand how to elaborate their information retrieval queries can also apply to helping them prepare messages that meet their needs and satisfy the protocols for communication with the Broker.

The Fast Workstation project has also implemented the ***Generic Description Generalizer***, a domain-independent facility for examining conceptual lattices to generate useful abstractions of a given concept description. This has been demonstrated in two domain-specific applications related to the project's concern with providing intelligent aids in the procurement of electronic parts. The first application, the ***Part Number Advisor***, is a small expert system that provides useful feedback to users who specify invalid part numbers; the system works by parsing the invalid part number, then using the ***Generic Description Generalizer*** to find valid close approximations to the properties implied by the invalid specification. The second application, the ***Equivalence Advisor***, accepts specifications of valid parts and finds alternative parts which are acceptable substitutes within given criteria. Implementation of these two applications, in addition to providing some useful application-specific functionality, has been served as an exercise relevant our research goals of developing techniques for enhanced modifiability and maintainability of software. The systems are the first demonstration of the approach of centering applications code around domain-independent interpreters (in this case, the ***Generic Description Generalizer***) of largely declarative specifications that are shared across multiple applications.

References

[Clancey 83] Clancey, W., "The Epistemology of a Rule-Based Expert System: A Framework for Explanation ," *Artificial Intelligence* 20, (3), 1983, 215-251.

[goldman 88] Goldman, N. and Miller, B., *Forms Int(.face Construction KIT Manual*, USC / Information Sciences Institute, 1988.

[Gurfield, et. al. 87] R.M. Gurfield, J. Postel, E. Anderson, P. Caruso, F. Chaparro, V. Morriss, A-L Neches, C.M. Rogers, & J. Brooks, FAST -- an Automated Broker for Electronic Parts Procurement, 1987. ISI Internal Working Paper.

[Kaczmarek 84] Kaczmarek, T., A Forms-Based Interaction Design Tool. Internal memo, USC-Information Sciences Institute

[Kaczmarek, et al. 86] Kaczmarek, T., Bates, R., and Robins, G., "Recent developments in NIKL," in *Proceedings of the National Conference on Artificial Intelligence*, pp. 978-985, American Association for Artificial Intelligence, Philadelphia, PA, 1986.

[Mac Gregor 87a] Robert Mac Gregor and Raymond Bates, The Loom Knowledge Representation Language, 1987.

[Mac Gregor 87b] Robert Mac Gregor, *The Knowledge Representation Project at ISI*, USC/Information Sciences Institute, Technical Report USC/ISI Technical Report RR-87-199, 1987.

[Mark 81] Mark, W., "Representation and Inference in the Consul System," in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, IJCAI, Vancouver, B.C., August 1981.

[Moser 83] M.G. Moser, "An Overview of NIKL, the New Implementation of KL-ONE," in *Research in Natural Language Understanding*, Bolt, Beranek, and Newman, Inc., Cambridge, MA, 1983. BBN Technical Report 5421.

[Neches 86] Neches, R., "Knowledge-based Interaction Tools," in *Proceedings of the 1986 IEEE Conference on Systems, Man, and Cybernetics*, The Institute of Electrical and Electronics Engineers, 1986.

[Neches 88] Neches, R., "Knowledge-Based Tools to Promote Shared Goals and Terminology Between Interface Designers," *IEEE Transactions On Office Information Systems*, In press 1988. Revised version of article in Proceedings of the First Conference on Computer-Supported Cooperative Work, December, 1986

[Neches, et al. 85] Neches, R., Swartout, W., and Moore, J., "Enhanced maintenance and explanation of expert systems through explicit models of their development," *Transactions On Software Engineering* SE-11, (11), November 1985, 1337-1351.

[Neches, Swartout, and Moore 85] Robert Neches, William R. Swartout, and Johanna Moore, "Explainable (and Maintainable) Expert Systems," in *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 382-389, International Joint Conferences on Artificial Intelligence and American Association for Artificial Intelligence, August 1985.

[Schmolze 83] Schmolze, J.G. & Lipkis, T.A., "Classification in the KL-ONE Knowledge Representation System," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 330-332, IJCAI, Karlsruhe, W. Germany, 1983.

[Swartout and Neches 86] Swartout, W. and Neches, R., "The Shifting Terminological Space: An Impediment to Evolvability," in *Proceedings of the National Conference on Artificial Intellingence*, 1986.

[Szekely 87] P.A. Szekely, *Separating the User Interface from the Functionality of Application Programs*, Ph.D. thesis, Carnegie-Mellon University, Computer Science Department, 1987.

[Tomovich 88] Tomovich, C., "MOSIS - A Gateway to Silicon," *Circuits & Devices*, March 1988.

[Tou, *et. al.* 82] Tou, F.N., Williams, M.D., Fikes, R., Henderson, A., & Malone, T., "RABBIT: An Intelligent Database Assistant," in *Proceedings of the National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, 1982.

[Yen 87] Yen, J. & Neches, R., Retrieval by Reformulation in a Multi-Purpose Browsing Interface, 1987. ISI Internal Working Paper.

[Yen 88] Yen, J., Neches, R., and DeBellis, M., "BACKBORD: Beyond Retrieval by Reformulation," in *Collected Papers of the Workshop on Architectures for Intelligent Interfaces: Elements and Prototypes*, pp. 219-235, ACM/SIGCHI, Asilomar, CA, 1988. (Also available as Technical Report *ISI/RS-88-202* from USC / Information Sciences Institute.)

[Yen, et al. 88] Yen, J., Neches, R., and DeBellis, M., "Specification by Reformulation: A Paradigm for Building Integrated User Support Environments," in *Proceedings of the National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, Minneapolis, MN, 1988.